AFRL-RI-RS-TR-2017-202

DESIGN METHODOLOGIES AND TOOLS FOR SINGLE-FLUX QUANTUM LOGIC CIRCUITS

UNIVERSITY OF SOUTHERN CALIFORNIA

OCTOBER 2017

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2017-202 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
JOSEPH OSMAN
Work Unit Manager

/ S /
JOHN D. MATYJAS
Technical Advisor, Computing
& Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
OCTOBER 2017	FINAL TECHNICAL REPORT	SEP 2015 – MAY 2017

4. TITLE AND SUBTITLE

DESIGN METHODOLOGIES AND TOOLS FOR SINGLE-FLUX QUANTUM LOGIC CIRCUITS

5a. CONTRACT NUMBER
FA8750-15-C-0203

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
N/A

6. AUTHOR(S)

Massoud Pedram, Coenrad Fourie

5d. PROJECT NUMBER
DMTS

5e. TASK NUMBER
US

5f. WORK UNIT NUMBER
CA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Southern California
3720 S. Flower St., CUB 303, MC 0701
Los Angeles, CA 90089-0701

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2017-202

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2017-5039
Date Cleared: 17 OCT 2017

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The goal of this project was to investigate the state-of-the-art in design and optimization of single-flux quantum (SFQ) logic circuits, e.g., RSFQ and ERSFQ and draw up a comprehensive research plan for developing a standard design methodology and supporting computer-aided design tools for the SFQ logic at the register-transfer-level and below. In the process, this project produced several preliminary, prototype software tools for proof-of-concept demonstrations, including an RSFQ cell library, a prototype standard cell timing characterization tool, a prototype static timing analysis tool, a prototype frontend logic synthesis tool, and a prototype backend place and route tool. The RSFQ library, and software tools can be accessed at http://sportlab.usc.edu/downloads/download-protected/. For username and password, please contact pedram@usc.edu.

15. SUBJECT TERMS
SFQ, RSFQ, ERSFQ, STA tool

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**JOSEPH OSMAN**
U	U	U	UU	38	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1. SUMMARY

The goal of this project was to investigate the state-of-the-art in design and optimization of single-flux quantum (SFQ) logic circuits, e.g., RSFQ and ERSFQ and draw up a comprehensive research plan for developing a standard design methodology and supporting computer-aided design tools for the SFQ logic at the register-transfer-level and below. In the process, this project produced several preliminary, prototype software tools for proof-of-concept demonstrations, including an RSFQ cell library, a prototype standard cell timing characterization tool, a prototype static timing analysis tool, a prototype frontend logic synthesis tool, and a prototype backend place and route tool. The RSFQ library, and software tools can be accessed at http://sportlab.usc.edu/downloads/download-protected/. For username and password, please contact pedram@usc.edu.

2. INTRODUCTION

To fulfill the main goal of this project, and develop a set of prototype design tool for SFQ design, we have broken down the problem into a number of tasks. The following list enumerates the tasks and subtasks of the proposed research.

1. Create a standard HDL-enabled cell library for RSFQ logic and characterize cells in the library (Cell library)
    a. Investigate and develop a generic HDL timing model of RSFQ circuits
    b. Characterize and release a standard cell library in HDL format
    c. Develop a library characterization prototype tool
    d. Validate and fine-tune the tool
2. Develop techniques and prototype tools for doing static timing analysis in RSFQ-based circuits (STA tool)
    a. Develop an STA technique for RSFQ circuits
    b. Develop an STA prototype tool
    c. Validate and fine-tune the tool
3. Develop logic synthesis techniques and a prototype tool targeting RSFQ-based circuits (Frontend tool)
    a. Investigate logic synthesis algorithms for RSFQ circuits
    b. Propose logic synthesis techniques for RSFQ circuits
    c. Develop a logic synthesis prototype tool
4. Develop techniques and a prototype tool for cell placement and clock tree design in RSFQ-based circuits (Backend tool)
    a. Investigate inter-connect delay models in RSFQ circuits
    b. Propose placement and clocking techniques for RSFQ circuits
    c. Develop placement and clock tree design prototype tool

Description of deliverables is as follows:

- M/D (month 9): Characterize and release a standard cell library in HDL format
- M/D (month 12): Develop a library characterization prototype tool
- M/D (month 12): Develop an STA prototype tool
- M/D (month 18): Develop a logic synthesis prototype tool
- M/D (month 18): Develop placement and clock tree design prototype tool

Description of each phase of the project, methods and algorithms developed is mentioned in Section 3. Results of the simulations on various benchmarks and circuits is discussed in Section 4. Finally, the report is concluded in Section 5.

3. METHODS, ASSUMPTIONS, AND PROCEDURES

Different methods and algorithms developed for standard cell library design, static timing analysis, logic synthesis, and physical design are discussed in this section.

3.1. Standard HDL-Enabled Cell Library for RSFQ Logic and Characterization of Cells

We developed a cell library for RSFQ logic in the MIT-LL SFQ5ee process. To arrive at a generic library in the short period allowed, some selections were made after due consideration:

1. Standard critical current density for Josephson junctions in a cell was set at 250 μA.

2. Cells were designed to be as fast as the process allows, so that we chose a slight under-damped Stewart-McCumber parameter $\beta_C = 2$ for all junction shunt resistors.

3. The cell library was designed to include a sufficient number of basic logic gates to allow complete synthesis: the NOT, 2-input AND, 2-input OR, and XOR gates.

4. SFQ-specific cells included are: the Josephson Transmission Line (JTL), Splitter, D Flip-Flop (DFF), Non-Destructive Readout Register (NDRO), Passive Transmission Line (PTL) Driver and PTL Receiver.

5. The DC-to-SFQ Converter (DCSFQ) was also included to allow simulations with non-SFQ input signals.

6. HDL modelling was done with Verilog.

7. Passive Transmission lines were fixed at approximately 5 Ω characteristic impedance.

Firstly, netlists representing the circuit schematics for each cell were designed for electrical/transient simulation in JSIM. These netlists were analyzed for operating margins via a margin analysis, and optimized through manual adjustment of low-margin parameters until circuit netlists with optimal margins were obtained.

HDL model generation is discussed in the subsections below.

For place and route tool design and verification, we required physical layout specifications for all cells, and example layouts of complete cells. Again, selections had to be made:

1. For row-based placement, all cells have equal height (the vertical layout dimension).

2. Cell width can vary between different logic cells, but must remain fixed for different layout versions of the same logic cell.

3. Routing in buried PTL layers uses a stripline configuration, with ground planes above and below the signal line.

4. PTL layers are accessed with vias surrounded by ground sleeves to minimize the impedance discontinuity. Ground planes above and below PTL signal lines are stitched together at arbitrary but user-definable intervals during interconnect synthesis. The via surrounds and ground plane stitches impose layout constraints, so that PTL track pitch was selected as 10 μm – leaving 5 μm for the line width (close to 5 Ω characteristic impedance) and 5 μm between lines to accommodate via ground sleeve and ground plane stitch objects.

5. All cell layout dimensions (height and width, as well as pin placement) must match the track pitch.

6. Bias pillars with ground plane vias for return current loop minimization are placed at specific locations in a layout.

7. Provision is made for a clock splitter at the top of each clocked cell, or a JTL clock propagation circuit at the top of each unclocked cell; but only for cells used with HL trees.

8. PTL drivers and receivers are built into cell layouts.

Generic layouts of the cells in the library were created as GDS files, as shown in Figure 1. Layout concepts and validity, as well as applicability to row-based layout synthesis were tested with example layouts, such as that of a 1-bit full adder shown in Figure 2 and Figure 3.

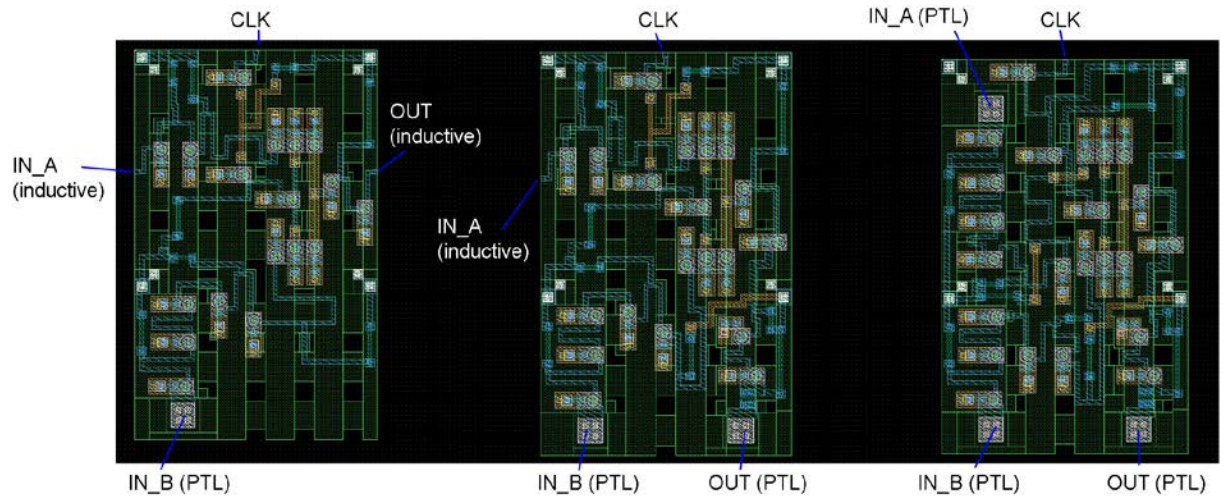

Figure 1. Different layout versions of the AND logic cell.

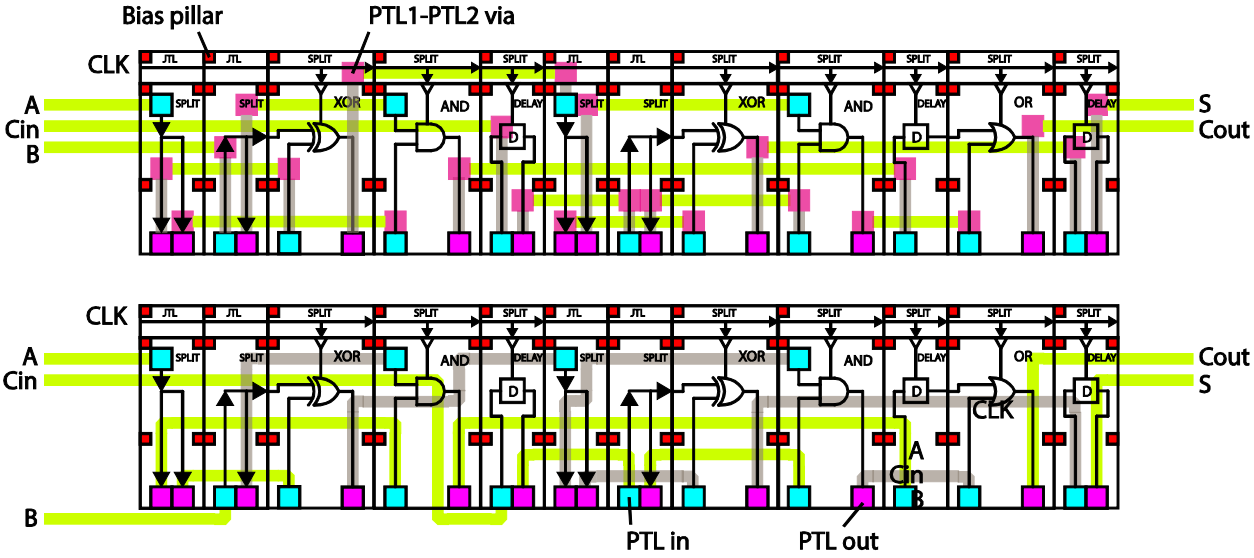

Figure 2. Schematic representation, true to scale, of 1-bit full adder in row-based placement, with different routing options.

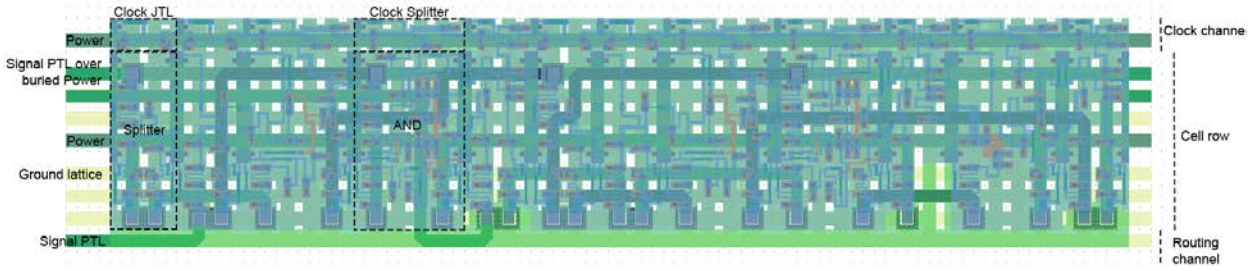

Figure 3. Actual layout of the 1-bit full adder to test layout concepts.

3.1.1. Investigate and develop a generic HDL timing model of RSFQ circuits

Although different timing models (such as setup and hold times) were considered, we eventually opted for a model with only two timing parameters that we presented earlier [1]: delay and critical timing. Here, delay timing describes the time from the arrival of an input pulse to the appearance of an associated output pulse. Critical timing describes the minimum time between the arrival of an input pulse to the arrival of a pulse at another input that would not violate the state transition.

Both timing parameters are state dependent, and are described as such in the extracted Verilog models.

3.1.2. Characterize and release a standard cell library in HDL format

A hand-crafted standard cell library in Verilog HDL format was developed, but this was superseded by the extracted library from the fully automated characterization tool described in section 3.1.3.

3.1.3. Develop a library characterization prototype tool

A cell library characterization tool was developed to automatically characterize cells and extract Verilog models. The tool, TimEx, operates as such:

1. A cell, or Device-Under-Test, is fed to TimEx along with a definition file. The definition file defines excitation methods, source, load and sink cells. These are then automatically combined with the input and output pins of the DUT to build a test bench.

2. The test bench is excited with all combinations of inputs, and states are searched.

3. States are detected by calculating the flux in every superconductive loop in the DUT. Josephson junction inductance is approximated as the static inductance, which as expected gives stable results where the more well-known small-signal equivalent inductance fails.

4. Each state is uniquely defined by its flux signature over all loops.

5. Once all states are known, each state is set up in turn, and all input combinations are probed. Output pulses are detected and delay times calculated.

6. For timing parameters, we follow Müller and Fourie's 2014 method [1], where critical times between any two input pulses are found through binary searches.

7. We added the provision that a delay (shift) in output pulses caused by input combinations approaching a state transition failure are also flagged as fail events. The allowable length of this delay is a variable parameter.

8. Pulse position is found from transient JSIM analyses by numerical integration over a sliding window. Window length and area detection threshold are user-definable.

9. Verilog model files are automatically constructed by TimEx to implement the timing characteristics and states extracted for the DUT.

10. A state diagram is generated for every extracted DUT to allow easy verification of the state dependencies.

11. Test bench files for both Verilog and JSIM are built automatically after extraction to allow easy verification and comparison of the extracted model to the original JSIM circuit.

The advantage of this method is that TimEx needs no information on the actual cell operation (such as the logic function). Roughly ten user parameters can be varied to control the limits of the extraction engine, should slower or unusual gates need to be extracted.

For the state of a cell to be investigated, the flux in every cycle (or loop / mesh) in the circuit must be evaluated. The flux is expected to be -1, 0 or 1 times the magnetic flux quantum (Φ_0). An algorithm thus finds all the meshes in the circuit so that cycle flux can be calculated from the sum of branch currents multiplied by element inductances (where the static estimation of Josephson junction inductance is used).

Resistive cycles would not store flux, so that any cycles that contain resistors are ignored. TimEx also disregards cycles that contain any input or output ports, as the assumption is that flux storage in the interconnect inductance between any load and input/output of the DUT represents unacceptable circuit behavior.

Numerical errors and errors in the Josephson inductances lead to cycle flux results that are spread over the range of roughly $0.9\Phi_0$ to $1.1\Phi_0$. TimEx then divides the cycle flux results by Φ_0 and rounds the values to the nearest integer (-1, 0 or 1), with the sign depending on the direction of flux through a cycle.

Even though analyses of time-based electrical simulations of SFQ circuits mostly rely on the phase evolution over Josephson junctions, TimEx investigates the voltage pulses at the inputs and outputs of the DUT. This is necessitated by the Verilog descriptions, which require response to the arrival of signals at the DUT inputs (which are typically inductors) and for which signal arrival times at the outputs need to be specified.

A simple way to detect pulses is to look for the peak values in voltage versus time plots, but this is risky. Firstly, a cell might fail to switch correctly and just create a ripple in the output of which peaks might be mistaken for pulses. Secondly, measurement of the pulses between the inductances of inputs/outputs to/from a load and the DUT often results in pulses that seem to oscillate, so that the peak value could shift by a few picoseconds depending on the inductances.

The voltage pulses transmitted between SFQ elements integrate to exactly one fluxon, so that a much more reliable way of detecting a pulse is to use a sliding numerical integrator. When the area

inside the sliding integrator is compared to a threshold (set as a fraction of Φ_0), the time at which the threshold is passed is a very stable way to characterize pulse arrival time.

In TimEx, the parameters SlidingIntegratorLength and PulseDetectThreshold set the length of the sliding integrator window (in time) and the integrated area threshold as a fraction of Φ_0 for pulse detection respectively (c.f. Figure 4).

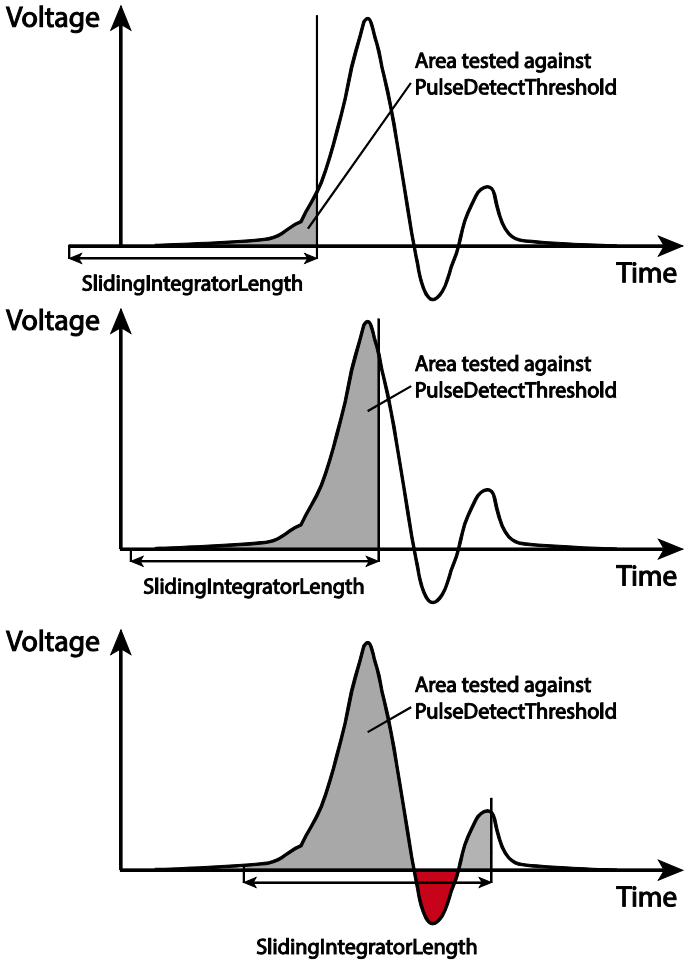

Figure 4. Pulse detection in TimEx with a sliding voltage-time integrator.

3.1.4. Validate and fine-tune the tool

The TimEx cell library characterization tool was validated through comparison of Verilog simulations of extracted models with equivalent transient electrical simulations in JSIM.

The tool worked very well from the start, with the only fine-tuning required being that of supporting exactly simultaneous inputs at two or more input pins.

An example extracted circuit, the RSFQ OR cell, is shown in as a Mealy state diagram in Figure 5, and in simulation in Figure 6 and Figure 7.

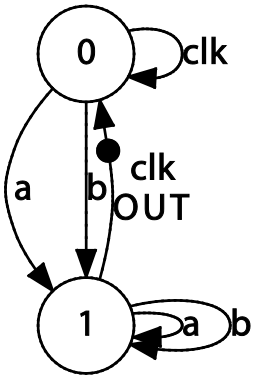

Figure 5. Mealy state diagram of extracted RSFQ OR cell.

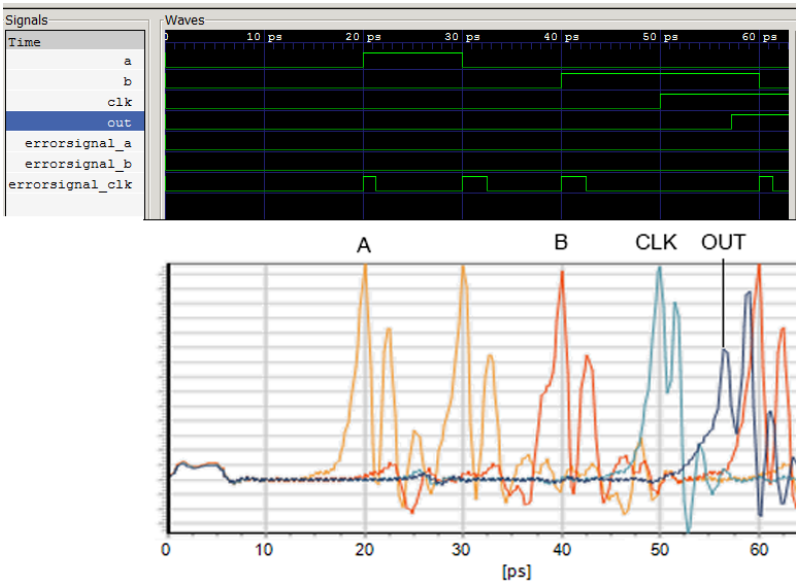

Figure 6. Verilog and JSIM simulations of the RSFQ OR cell, which shows that pulse positions match.

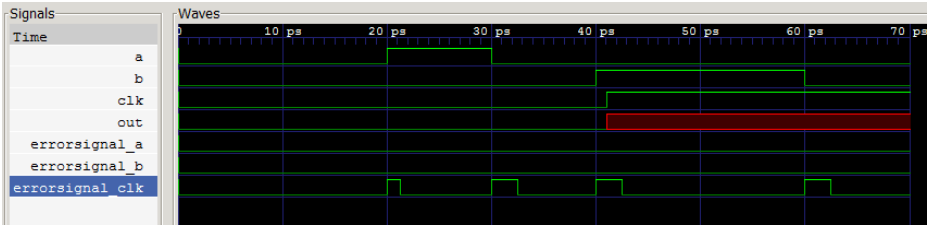

Figure 7. Verilog simulation of RSFQ OR cell, extracted with TimEx, where a timing violation occurs.

```
always @(posedge a or negedge a) // execute at positive and negative edges of input
   begin
      if ($time>4) // arbitrary steady-state time)
         begin
            if (errorsignal_a == 1'b1)  // A critical timing is active for this input
               begin
                  outfile = $fopen("errors.txt", "a");
                  $fdisplay(outfile, "Violation of timing in module %m; %0d ps.\n", $stime);
                  $fclose(outfile);
                  out <= 1'bX;  // Set all outputs to unknown
               end
            if (errorsignal_a == 0)
               begin
                  if (cell_state == 0)
                     begin
                        cell_state <= 1;
                     end
                  if (cell_state == 1)
                     begin
                        errorsignal_b = 1;  // Critical timing on this input; assign immediately
                        errorsignal_b <= #(ct_state1_a_b) 0;  // Clear error signal
                        errorsignal_clk = 1;  // Critical timing on this input; assign imm.
                        errorsignal_clk <= #(ct_state1_a_clk) 0;
                     end
                  if (cell_state == 2)
                     begin
                        cell_state <= 3;
                     end
                  if (cell_state == 3)
                     begin
                        outfile = $fopen("errors.txt", "a");
                        $fdisplay(outfile, "Illegal input”);
                        $fclose(outfile);
                        out <= 1'bX;  // Set all outputs to unknown
                     end
               end
         end
   end
```

Figure 8. Excerpt from RSFQ AND gate Verilog structure extracted with first implementation of TimEx.

A change in the way that Verilog models are built from cell extraction has been made to avoid simulation errors when two inputs arrive in the exact same time step. Our earlier models used non-blocking assignment of a change in cell state in response to inputs (see Figure 8), which caused simultaneous inputs to miss the state change effected by each other. Our solution is a blocking assignment of state change, and a case-endcase block to prevent the immediate assignment of a state change from triggering other unwanted events, as shown in Figure 9.

```
always @(posedge a or negedge a) // execute at positive and negative edges of input
   begin
      if ($time>4) // arbitrary steady-state time)
         begin
            if (errorsignal_a == 1'b1)  // A critical timing is active for this input
               begin
                  outfile = $fopen("errors.txt", "a");
                  $fdisplay(outfile, "Violation of timing in module %m; %0d ps.\n", $stime);
                  $fclose(outfile);
                  out <= 1'bX;  // Set all outputs to unknown
               end
            if (errorsignal_a == 0)
               begin
                  case (cell_state)
                     0: begin
                           cell_state = 1;  // Blocking statement -- immediately
                                       end
                                 1: begin
                           errorsignal_b = 1;  // Critical timing on this input; assign imm.
                           errorsignal_b <= #(ct_state1_a_b) 0;
                           errorsignal_clk = 1;  // Critical timing on this input; assign imm.
                           errorsignal_clk <= #(ct_state1_a_clk) 0;
                        end
                     2: begin
                           cell_state = 3;
                        end
                     3: begin
                           outfile = $fopen("errors.txt", "a");
                           $fdisplay(outfile, "Illegal input”);
                           $fclose(outfile);
                           out <= 1'bX;  // Set all outputs to unknown
                        end
                  endcase
               end
         end
   end
```

Figure 9. Excerpt from RSFQ AND gate Verilog structure extracted with improved version of TimEx.

3.2. Techniques and Tools for Static Timing Analysis In RSFQ-Based Circuits

A Static Timing Analysis tool for RSFQ circuit designs was developed and tested using 4, 16 and 32-bit Kogge-Stone adders. The tool is written in C++ and is cross-platform. We utilize the timing information from the HDL library generated by the RSFQ characterization tool TimEx. The tool produces clocking information as well as path statistics for the DUT.

3.2.1. Develop an STA technique for RSFQ circuits

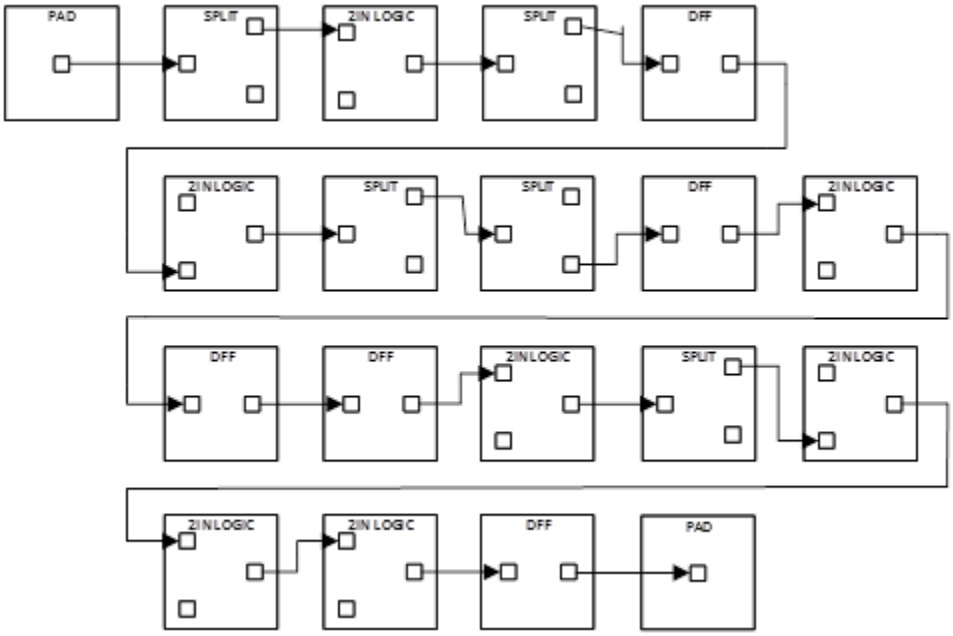

Figure 10. Path identified in a 16-bit Kogge-Stone adder using BFS.

RSFQ logic is synchronous and therefore needs to be clocked for the pulse to propagate to the next gate. Initially we opted to identify the slowest gate-to-gate delay within the DUT and present this as the system clock. This however proved to be an insufficient solution and further investigation was done. The DUT needed to have a global clock at which the design would produce an output for every clock pulse. To identify what this global clock speed needs to be, path identification using a breadth first search (BFS) algorithm was employed. This algorithm finds all the paths from input to any possible output in the design considering the branching that occurs when a splitter cell is encountered. An example of a path identified using this algorithm is seen in Figure 10. Using this algorithm, we are able to identify the critical path through the design. The critical path delay (t_{crit}) is the summation of delays through all the logic and splitter cells in the critical path. The global clock is then $1/t_{crit}$ and is the maximum speed the design would produce and output after every clock pulse.

To identify the maximum system clock frequency with the DUT, further care is needed with regards to what type of clocking scheme is employed by the design. For simple concurrent clocking, the initial hypothesis would hold true and it would simply be the slowest gate-to-gate delay. If, however H-tree or the hybrid HL-tree clocking scheme is used the system clock needs to be determined using other means.

For either of the clocking schemes the DUT clock tree needs to first be rebuilt and path analyzed to identify what form of clocking is used and how to approach the analysis. In the case of an H-tree clocking scheme the fastest system clock that would produce no timing violations would be the deepest branch from clock entry to any logic cell in the design assuming the H-tree is balanced. For the HL-tree clocking scheme we need to find the shallowest non-clock tree related logic branch and identify this as the t_{tree_min}, from this point in any of the branches we determine the maximum

delay from t_{tree_min} depth through any logic cell summing the delays of any splitters along the way. The system clock is then the maximum speed at which the clock within the DUT can propagate without causing timing violations in any of the cells.

3.2.2. Develop an STA prototype tool

We present the design flow for a prototype STA tool for RSFQ-based circuits in Figure 11 and Figure 12. In our first implementation, the only input file that we could analyze was a JSIM/SPICE netlist file which had no apparent clocking scheme and would simply do a path identification by using the SPICE nodes and would calculate the critical delay by summing the clock to output delays for every cell in every path and finding the largest delay. This is of course a rough first approximation, which produces an inaccurate global clock. As the project progressed, and the post place and route output format crystallized, the STA concepts could be refined and the tool improved.

The second input file that was introduced as DUT for the STA prototype tool, as a result of the abovementioned extension of the tool, was a Cadence DEF file which is produced by the RSFQ place and routing tool. This file contains a lot more information about the design including the clocking scheme as well as the wire lengths and via locations between every gate. This allows us to perform a much more accurate analysis of the DUT and in terms of the global clock as well as the system clock.

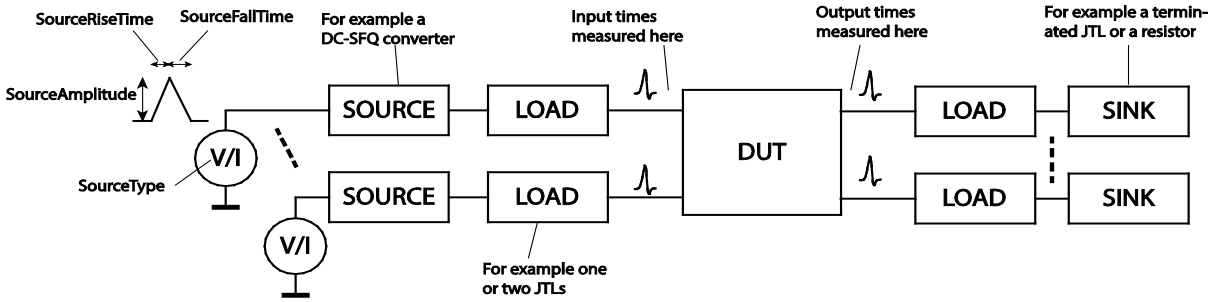

Figure 11. Schematic diagram to describe test bench setup around the Device-Under-Test in Timex.

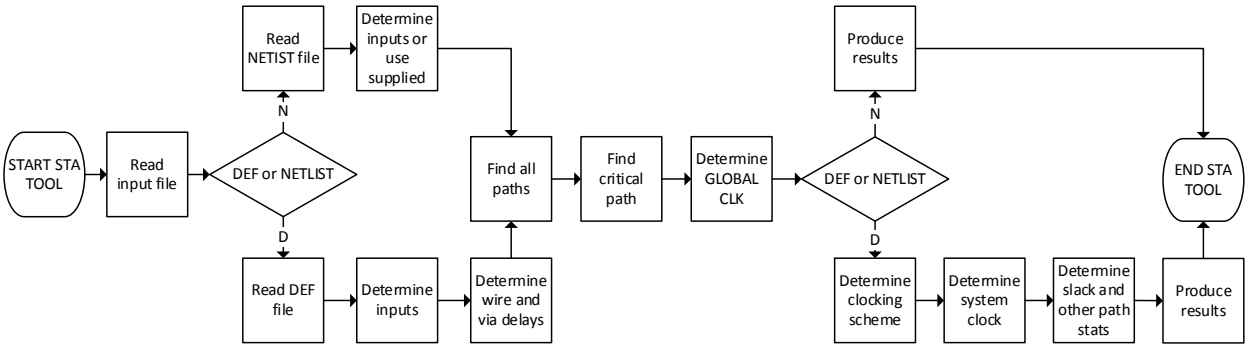

Figure 12. Prototype STA tool for RSFQ-based designs.

```
File specified to analyse: "16bitksa_route.def"
Critical path time: 5.54009e-10s or rather 554.009ps
Global clock: 1.80502GHz
Slack: -2.54009e-10s
Total slack: -3.17481e-07s
Mean path time: 4.58558e-10s or rather 458.558ps
Path time variance: 1.49185e-21s or rather 1.49185e-09ps
Path time standard deviation: 3.86245e-11s or rather 38.6245ps
H-Tree critical time: 73.6836ps
H-tree clock: 13.5715GHz
Longest NET: xor2a_104_n104 -> DFF_438_n1686 => 2790 micron
HL-Tree critical time: 100.7ps
HL-tree clock: 9.93049GHz
```

Figure 13. Output snippet from the STA tool.

Further timing information is also presented along with the clock speeds for the DUT. Among this information is the mean path time, path time variance as well as the standard deviation of the path times. It also enables the designer to input a target time for the design which produces a slack value. Positive slack meaning that the critical time is less than the target time and the design meets the speed requirements set by the designer. A negative slack value indicates that the design does not meet the speed requirements and a total slack value is produced which can be seen as a measure of how badly the design misses the mark.

3.2.3. Validate and fine-tune the tool

The STA tool was tested with 8, 16 and 32-bit designs in both JSIM/SPICE netlist format as well as DEF file format. The results with the 16-bit Kogge-Stone Adder suggest a maximum serial speed of only 1.8 GHz as seen in the output snippet from the STA tool in Figure 13, assuming of course that there is no pipelining. The maximum system clock when using the hybrid HL-tree method is close to 10 GHz.

3.3. Logic Synthesis Algorithms for RSFQ Circuits

We use ABC [2] for logic synthesis. ABC is a software program written in C for synthesis and verification of binary sequential logic circuits appearing in synchronous designs [2]. We have used some of the existing features of ABC and added some new features so that an RSFQ circuit can be synthesized using this tool.

3.3.1. Specifications of Cell Library and Input Design

Inputs to ABC include (i) a Verilog (or BLIF [3]) description of a standard CMOS circuit, and (ii) a cell library, which is a list of all available gates along with their Boolean function, area, and delay. ABC then synthesizes the netlist and maps it to the cells specified in the cell library such that the circuit delay is minimized.

3.3.2. Path-Balancing

RSFQ circuits are gate-level pipelined. Hence, every logic gate (cell) requires a clock signal to process the data. In such a circuit, to ensure that the input data arrives at each cell at the correct clock cycle, the circuit must be path-balanced. Path-balancing ensures that all paths in a circuit from any primary input to any primary output have the same number of clocked cells (logical depth). DFFs should be inserted between appropriate cells to path-balance a circuit. Using ABC, we initially insert DFFs between any two consecutive cells, where the number of inserted DFFs is equal to the difference between the logic levels of the corresponding cells minus one. Next, we run the standard retiming algorithm in ABC to minimize the DFF (register) count in a sequential circuit. At this step, we have a fully path-balanced sequential circuit with minimum number of DFFs.

3.3.3. Splitter-Insertion for Fanouts

In the synthesized circuit, there will be many fanouts which are greater than one. After path-balancing, we visit each gate of the circuit and if we find a fanout greater than one, we insert splitters using a balanced binary tree structure to minimize the latency through splitters. However, large fanouts are not desirable in RSFQ technology, since large fanouts not only complicate placement and routing, but also decrease the clock frequency as delay of splitting signals increases.

Accordingly, we are investigating synthesis methods to produce circuits with small fanouts. The main research conducted could be sub-divided into following categories:

1. Simulation of RSFQ cells and circuits and the Generation of RSFQ Library:
   We spent lot of time trying to understand the concepts of RSFQ and the availability of several gate circuits for the project usage. We simulated all the basic gates using the JSIM simulator and created a cell-library with our own JSIM netlists. The synthesis of circuits is done using our self-generated gates.

2. Design of Complex gates:
   A circuit with a large number of logic levels (higher depth), typically requires more path-balancing DFFs (PB-DFFs), which subsequently increases the area and power consumption of the circuit. Logical depth also indicates the latency of the circuit. This implies that if we can design a single logic cell to realize a relatively complex Boolean function, using this single cell instead of implementing it with basic 2-input cells, not only reduces the total cell count, but also decreases the logical depth of the circuit. Thus, using complex cells is helpful in reducing the area, power consumption, and latency of RSFQ circuits. For this purpose, we designed 3, 4, 5-input AND and OR gates and a special 3-input cell to implement A + BC (AND-OR), which is widely used in the *carry look-ahead adder* (CLAs). Accordingly, our A + BC cell generates the output in one clock stage with a clock-to-Q delay comparable to that of a 2-input AND cell, while using 2 input gates, it requires 2 clock cycles to produce the output. Schematics of 3-input AND and OR cells and the corresponding margins are shown in Figure 14.

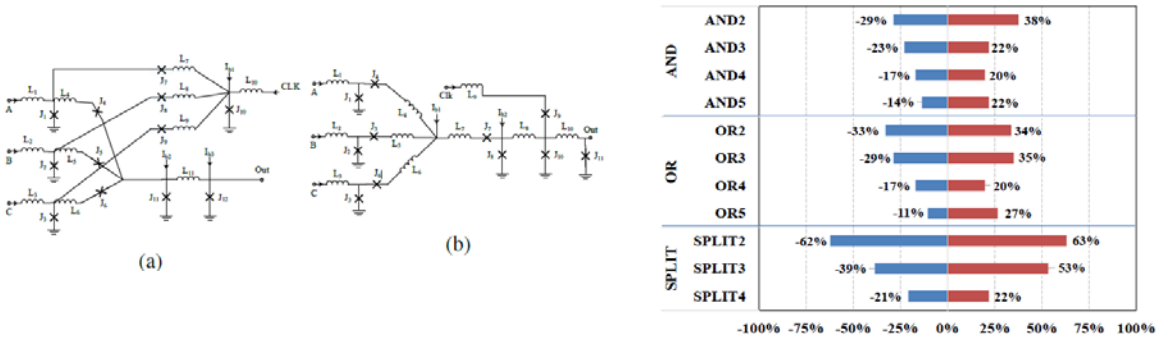

Figure 14. (a) 3-input AND gate (b) 3-input OR-gate (c) Critical margins of our complex gates.

3. Integer divider circuit Generation

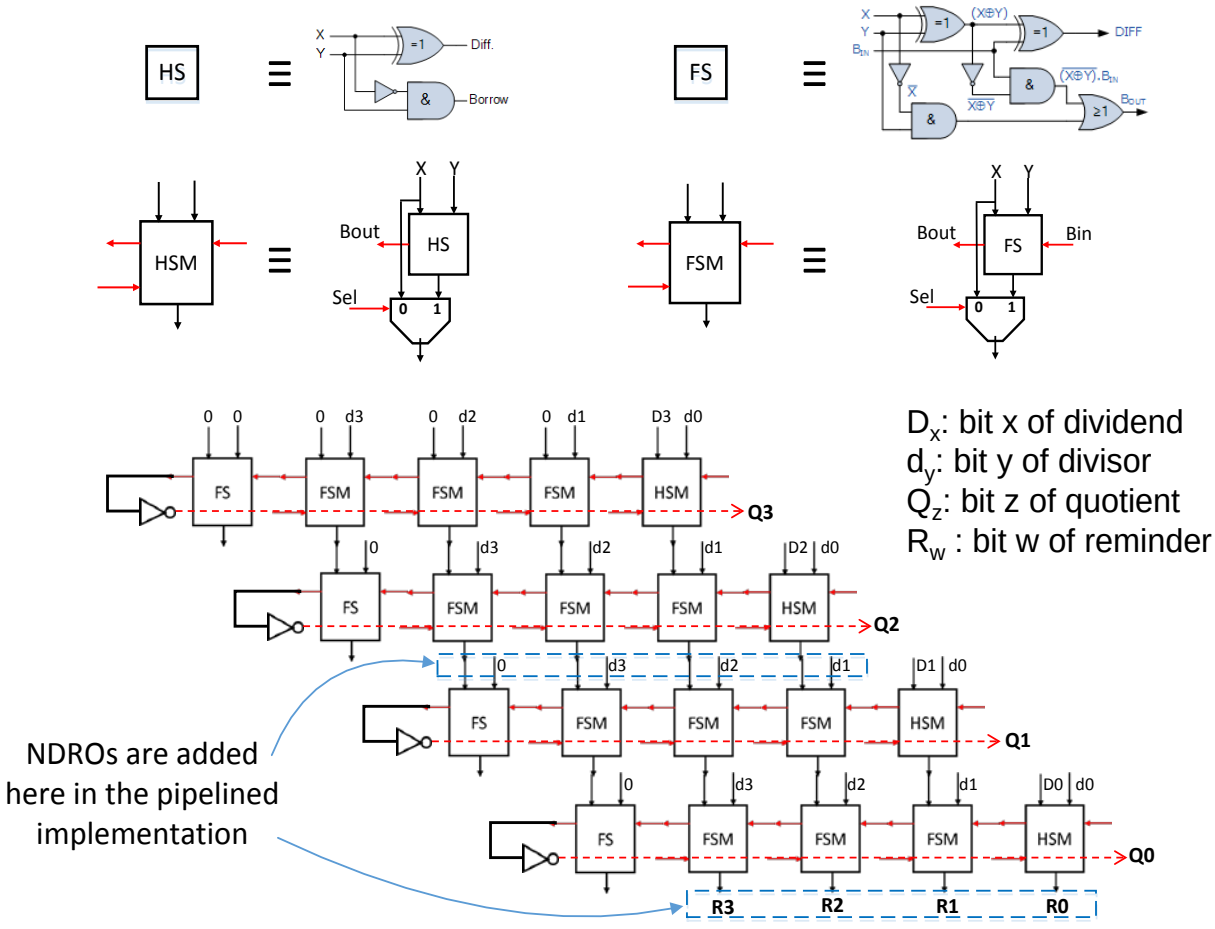

Figure 15. Integer divider circuit implementation.

The basic implementation of an integer divider circuit is shown in Figure 15. Modifications were done to the below structure to reduce the total number of gates required for the implementation of the integer divider.

We came up with several designs to optimize the area and speed of the integer divider circuit. One such implementation has a slow clock and a fast clock. The slow clock serves as the clock signal for *Non Destructive Read Out* DFFS (NDROs), which separate different stages of logic (pipeline DFFS). The fast clock is used for the rest of the gates in the circuit. Another design is a bit-serial design which has only one row of logic. To implement the bit-serial integer divider, we used two shift registers: One to perform the load-and-shift operation, and the other for the save-and-shift operation. For the load-and-shift operation, we used mergers in between DFF cells. All data bits are loaded into the merge block inputs and subsequently stored in the corresponding DFFs. We used the slow clock to operate these DFFs. At each cycle, the concerned data bit gets into the first HSM (Half Subtractor with Multiplexer) cell of the bit-serial divider structure, moving from the most significant bit to the least significant bit. For the save-and-shift operation, we used splitters instead of mergers and the data movement is in opposite direction of the load-and-shift operation. Figure 16 and Figure 17 depict the basic operation of the bit-serial implementation.

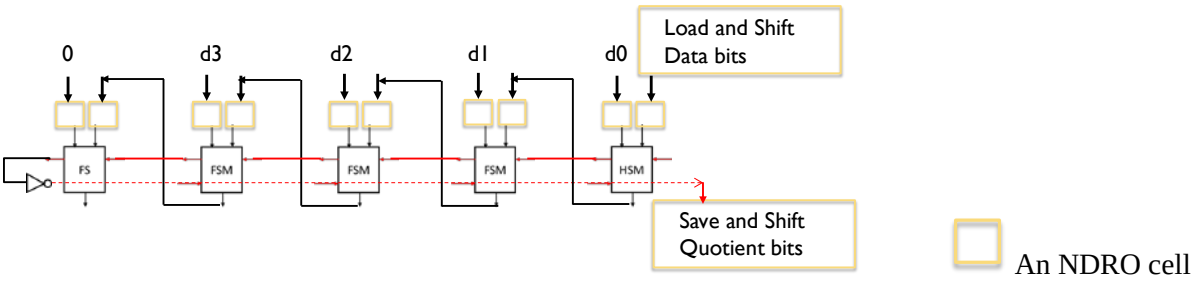

Figure 16. Bit-serial implementation of the 4-bit integer divider. HSM: Half Subtractor with Multiplexer; FS: Full Subtractor; FSM: FS with Multiplexer.

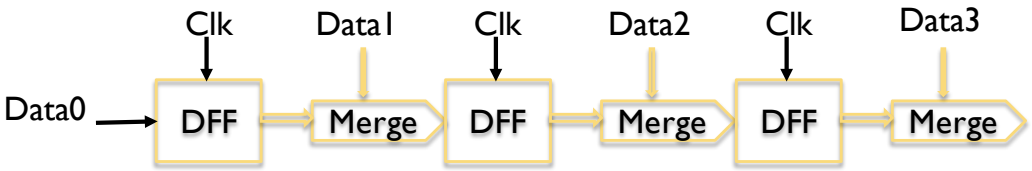

Figure 17. Shift register for the load-and-shift operation.

4. Design of multiple fanout circuits for clock-tree network

Rapid Single Flux Quantum (RSFQ) logic cells have traditionally been limited to driving one fanout cell only, due to complications in distributing the single flux quantum pulse to multiple fanouts. However, we presented a method to modify the interface of standard RSFQ cells in order to support multiple fanouts. For clock distribution, having multiple fanout drive capability is very important as the RSFQ logic is gate-level pipelined and requires clock for every logic operation. In general, the clock signal is split using splitter cells to provide the signal to different cells in the same logic circuit. To support multiple fanouts without using splitter cells, the basic idea is to connect the output of one splitter to more than one gate, and compensate the input

current reduction by increasing the bias current of the Josephson junctions at the receiving end of the fanout. This helps simplify the clock routing process and reduces area usage but it also tends to decrease the circuit margins. However, we show that the yield is not compromised by our proposed technique and thus we present an algorithm for modifying the interface of RSFQ logic cells for this purpose. Details of the implementation could be found in [4].

3.4. Cell Placement, Clock Tree Design and Routing in RSFQ-Based Circuits

This part describes several prototype software tools for synthesis and physical design of single-flux quantum (SFQ) logic circuits, including a standard cell characterization tool, a static timing analysis tool, a frontend synthesis, and a backend placement and routing tool. The overall flow is shown in Figure 18.

The tool suite used for logic synthesis, placement, clock tree synthesis and routing, called RSFQ_Mapper, receives as an input a high-level description of the design in Verilog HDL or BLIF formats and maps it to a RSFQ chip.

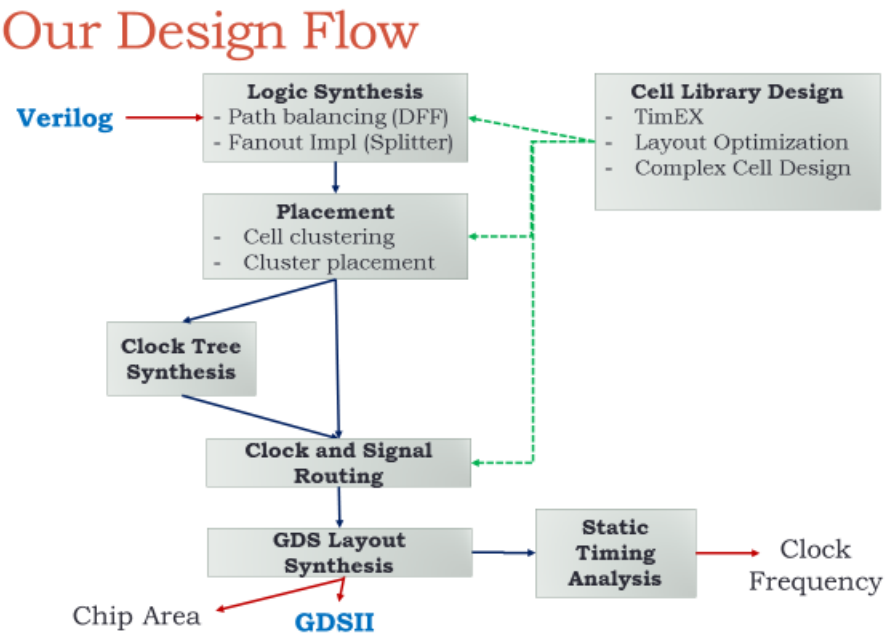

Figure 18. Overall design flow.

The mapping process is comprised of the following tools:

1. A logic synthesis tool, based on ABC with proper modifications and new features to generate RSFQ-compatible netlists.
2. A placement tool, which is written for RSFQ circuits, and uses the following tools:
    a. A global placement algorithm to generate global ordering for all the cells while reducing the total wirelength
    b. A Clock tree synthesis tool called BST/DME: This tool is used to find the locations of clock splitters for constructing the H-tree clock network.

c. A Detailed placement tool to further refine the placement solution.
d. A legalization algorithm to remove cell overlaps and produce final legal solution.

3. Routing tool based on a maze router algorithm using an open-source tool (Qrouter).

The final output of the "RSFQ Mapper" is a placed-and-routed netlist in Design Exchange Format (DEF). The mapper can also draw the final layout of the chip using Magic. Each part of RSFQ_Mapper is explained in details as follows:

3.4.1. Proposed Placement and Clock Tree Synthesis Tool

The outcome of logic synthesis tool, is a netlist consisting of cells and their connection. Based on the total area of the cells and total number of I/O pads, the width and height of the chip is calculated. Once the chip floorplan and position of I/O pads is determined, cells should be placed on the chip such that total wirelength of the nets is minimized. Proposed placement flow is shown in Figure 19. Overall placement flow.

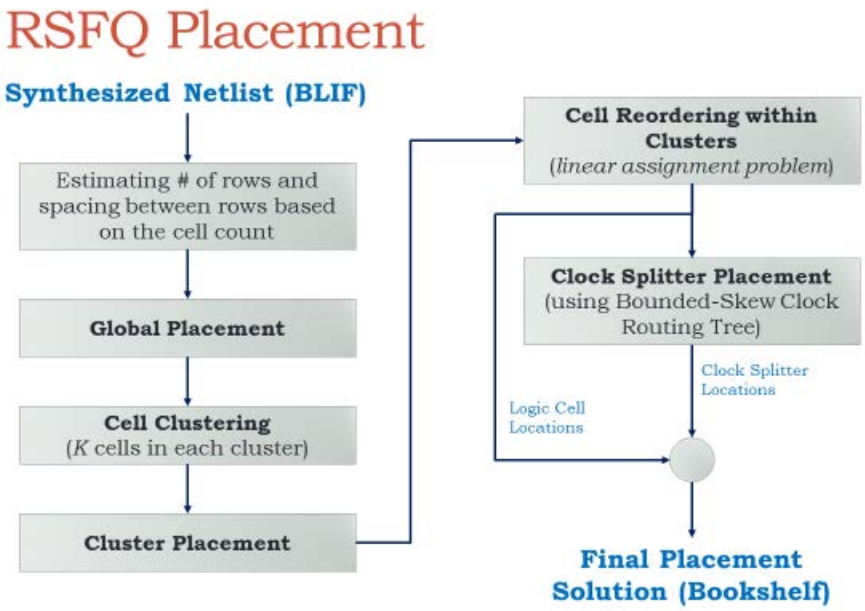

Figure 19. Overall placement flow.

First, a global placement algorithm is performed which distributes the cells in the chip. We use SimPL [5], one of the state-of-the-art placement algorithms that is fast and multiple objectives can be easily integrated within its flow for the global placement step. The global placement algorithm is described in [5]. It starts with an initial uniform placement which distributes cells randomly in the chip area. Based on this initial placement, and Bound to Bound net model [6], a system of linear equations is formed, and solved to produce new locations for all the cells, such that total wirelength, which can be modeled by the half-perimeter wirelength (HPWL), is minimized. This step is repeated 5-7 times, until total wirelength improvement converges. This step reduces the wirelength significantly but leads to a lot of overlap in the middle of the chip.

Consequently, to remove the overlaps and produce a legalized solution while maintaining the solution quality, the Look Ahead Legalization (LAL) algorithm is performed. This algorithm removes the overlap by placing pseudo anchors to create expansion forces and discourage overlaps. Once the expansion forces are formed, the system of linear equations is updated and solved to produce new positions. This step is repeated several times (at most 55 times), and at each step total HPWL is increased because of legalization.

We have implemented the SimPL placement algorithm using C++. Once the global ordering of the cells is determined, all cells are legalized to remove any potential overlaps and a detailed placement algorithm is run to further improve the placement quality. Although the outcome of this step could be directly used for clock tree synthesis and routing, this may lead to a large cell area and degraded performance. In current RSFQ technology all the cells have a fan-out of 1 and to propagate the signals to more than one fan-out, splitters should be used. Thus, a complete H-tree with n clock sink nodes needs n-1 clock splitters (assuming $n = 2^m$) which leads to a large area dedicated to clock tree network. Furthermore, almost all the cells in the SFQ netlist need a clock signal, whereas in CMOS designs, 15-20% of the cells receive clock signals. Moreover, the total number of metal layers in current technology is limited. The above issues motivate a clock tree aware approach to reduce the size of the clock tree network which leads to lower total chip area and higher clock frequency, as final frequency is a function of longest path and hence chip area.

To reduce the area dedicated to the clock network, cells of the same logic level are clustered and a clock signal is propagated to each cell group, rather than each individual cell. This clocking scheme, called an HL-tree clock network, is shown in Figure 20. The top portion of each cell which includes a splitter and JTL is used to propagate a clock signal locally to the other cells in a cell group (super-cell).

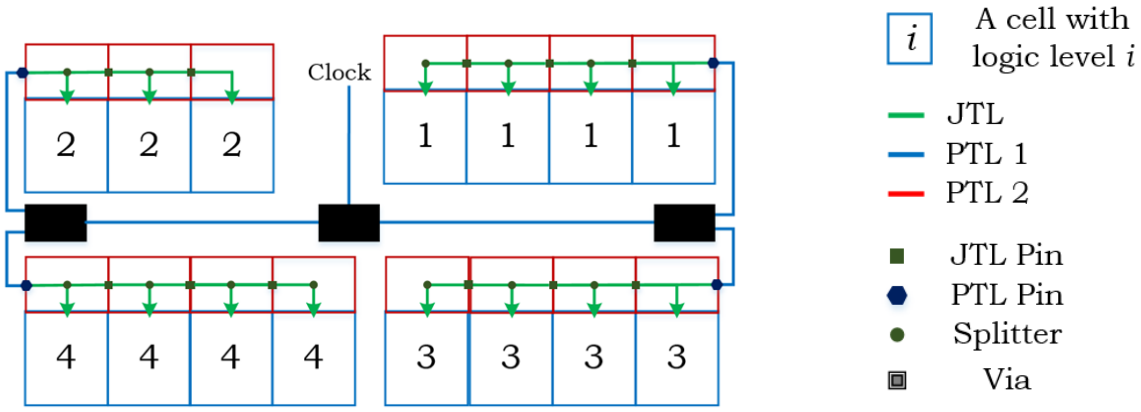

Figure 20. HL-tree clock network with a group size of 4.

After global placement, legalization, and detailed placement, we examine each cell row one at a time. In each cell row we start from the left side and move toward the right hand side, creating groups of cells such that each cell group has at most k cells with the exact same logic level. Each

such cell is called a super-cell. These super-cells are then placed in the same row based on the initial ordering of the first cell added into the cell group. Finally, the clock sink nodes are determined, and the clock tree synthesis engine propagates the clock signal to all clock sinks using a zero-skew tree (BST/DME algorithm [7]). BST is implemented in two phases. A bottom-up phase constructs a binary tree of merging regions which represent the loci of possible embedding points of the internal nodes, and a top-down phase determines the exact locations of the internal nodes [7].

Once the clock tree is built, splitter cells are inserted into the routing channel to propagate the clock signal from the source to all the sinks. The placement tool receives the input netlist in the Bookshelf format [8] and a variable, which serves as an upper bound on the cell group sizes (the default value is set to one, which means that no cell grouping will be done and the initial placement is used, resulting in a complete H-tree clock network.)

Furthermore, to improve the clock routing and decrease the total wirelength dedicated to clock routing, 8 different instances of clock splitter cell are used. These vary in the postiton of the driver and the two fan-outs of each clock splitter cell. The different implementations of clock splitter cells are shown in Figure 21.

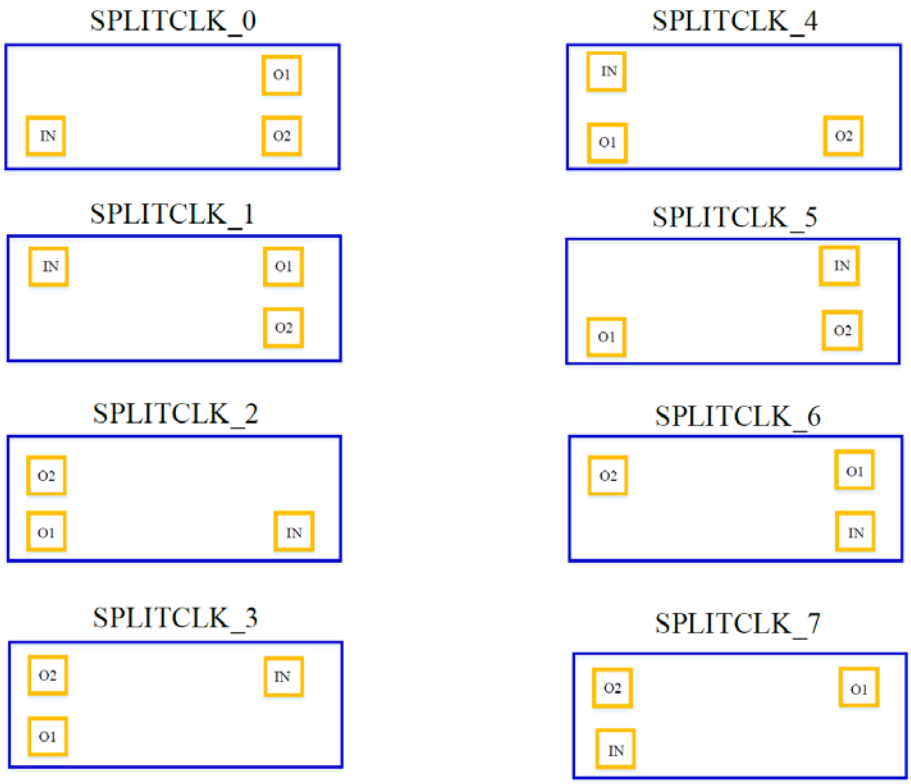

Figure 21. 8 different instances of clock splitter cells.

Based on the location of driver and fan-outs, one of the 8 splitters is chosen and placed as part of the clock tree. This approach was tested on a 32bit Kogge-Stone adder with 2415 nets and 2852

clock splitter cells, which results in a 9% reduction in total wirelength of clock nets and a16% reduction in max via count. After the placement and clock tree synthesis, the placed netlist (including cells, I/O pins, and clock tree splitters) is passed to the routing tool to route clock and signal nets.

3.4.2. Proposed Routing Tool

Given a placed netlist and technology information, the routing problem is to determine the necessary wiring, including net topologies and specific routing segments, to connect the pins of all the cells while meeting the design rule constraints and the routing resource capacities. The inputs to the routing program, Qrouter [9], are given in open standard *library exchange format* (LEF) and *design exchange format* (DEF) files, which together represent the complete physical layout of an integrated circuit in an ASCII format. Specifically, LEF includes design rules and abstract information about the cells, whereas DEF represents the netlist and circuit layout. Both files are compatible with most fabrication processes as most vendors supply standard cell definition in a compatible LEF file readable by most EDA layout tools with the detailed routing geometry.

Like the placement problem, routing of signal nets is performed in two steps. (i) G*lobal routing,* whereby wire segments are tentatively assigned to coarse-grain routing regions and (ii) *Detailed routing,* whereby specific routing tracks, vias, and appropriate segments of metal layers are assigned to each net in a manner that is consistent with the given global route of the net in question. Accordingly, the detailed router must account for design rules. Global routing algorithms can be classified into two approaches: (i) concurrent and (ii) sequential [10]. The first approach is not suitable for large circuit routing problems, because it results in a large integer linear programming problem (which is an NP-hard optimization problem) without a polynomial time solution. The latter approach may be further classified as restricted or general purpose routing. The general-purpose algorithm consists of maze router and line-search algorithms. The maze router with the shortest path guarantee is based on Lee's algorithm and can find all the existing paths among the grids without exceeding the constrained cost function. The line-search algorithm cannot promise the shortest path but can be significantly more efficient [11]. Since the delay after *Josephson transmission line* (JTL) transmission is comparable to the gate delay, the shortest path should be guaranteed by the routing algorithm in the worst case.

In current designs, we do not use JTLs for signal routing, since (i) JTLs require JJs and hence occupy the active layer, which in turn complicates the placement tool, and (ii) JTLs are slower than PTLs especially for long-distance communications. Therefore, signal routing is done either by direct connection if the source and destination pins are abutted or by PTLs otherwise. Accordingly, we will have different templates for the logic part of the standard cells. As an example, Figure 22 shows a template for a 2-input gate.

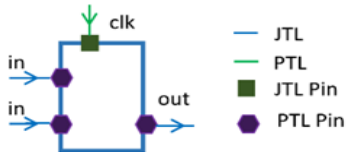

Figure 22. A template for the logic part of a standard cell that implements a 2-input Boolean gate.

Our routing tool will be built on top of the open-source Qrouter tool [9], which is developed based on the standard Lee's algorithm. Given LEF and DEF files, the routing area is partitioned into a 2-dimensional grid of routing tracks. A wave propagation method connects the identified source and target nodes while avoiding obstructions during propagation and calculates the corresponding cost along multiple paths. The L-shaped or doglegged paths with the lowest cost are then traced back to the source node from the target and committed to memory, as shown in Figure 23.

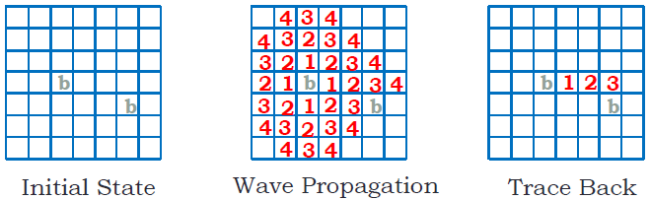

Figure 23. A wave propagation method of Lee's maze router algorithm.

The grid positions occupied by routed paths become obstructions for future routes on other nets (or become additional source or target nodes for further routing of the same net). There are two steps to complete the routing of a circuit. After sorting nets in the order determined by the longest Manhattan distance of two nodes in a net, the first step seeks to find the routing solution for each net one at a time without exceeding a cost function determined by the segment cost times the distance, while keeping track of any failed nets in a netlist [12]. In the second step, each net that has failed is routed again, allowing the net to create electrical shorts with other routed nets. At the same time, the routing cost bound for the said net is increased exponentially to maximize the chance of routing success. Subsequently all such newly shorted nets are removed and added to the list of failed routes, to be re-routed at a later time as shown in Figure 24.

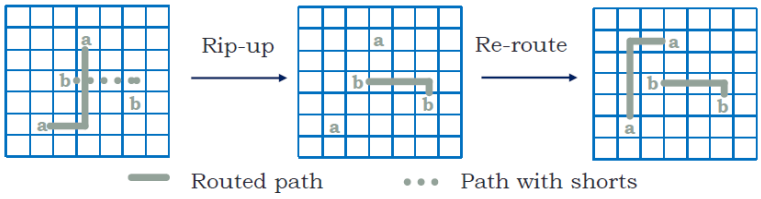

Figure 24. Rip-up-and-re-route process of two nets, (a-a) and (b-b).

This process continues until all nets have been routed. To avoid an infinite loop in this iterative rip-up-and-re-route process, the same sequence of rip-up and re-route for two nets is not allowed to happen more than once.

Table 1. Routing Cost Parameters.

ROUTING COST	VALUE
SEGMENT COST	1
JOG COST	20
VIA COST	25
CROSSOVER COST	1

Table 1 shows the details of routing cost parameters. Segment cost determines the cost of routing a segment in the preferred direction; jog cost denotes the additional cost of one routing segment in the direction other than the preferred one; via cost shows the cost of going from the metal layer i to metal layers (i+1) or (i-1). Crossover cost is the cost of routing directly over or under an unrouted pin connection to a cell.The segment cost was set by the minimun integer as a reference value. High jog cost and via cost were used to ensure that the Qrouter chooses a detour path on a single metal layer instead of one with multiple vias and different metal layers.

Figure 25 shows cell routing transparency of a 2-input AND gate given four routing metal layers, which in turn defines over-the-cell routing capacities in each routing direction. We defined signal ports of inputs and outputs in the bottom metal layer (metal 1) and clock ports of inputs and outputs in the top metal layer (metal 4). The bias pillars at the four corners of the cell obstruct routing on all layers. The design rule constraints require that the minimum distance between the centers of two wires is one pitch (10 µm). The routing capacities of the bottom layer over the cell for Qrouter thus are confined by the signal ports of both inputs and outputs. A similar condition applies to the top layer, this time by clock ports, but there are more allowable routes over cells of other routing layers in between. Consequently, the Qrouter prefers routing long-distance nets by utilizing middle metal layers because of their higher flexibility.

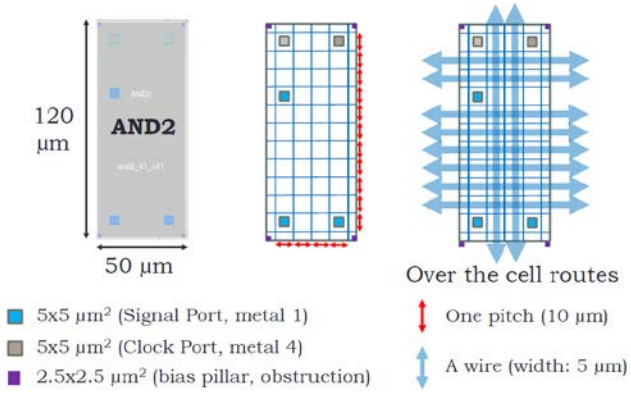

Figure 25. Cell Routing Transparency of a 2-input AND gate.

We developed a parser in C language to transform the bookshelf format [8] into DEF/LEF format as input files and designed a feedback routing system integrating Qrouter to find a complete routing layout for a VLSI circuit. See Figure 26.

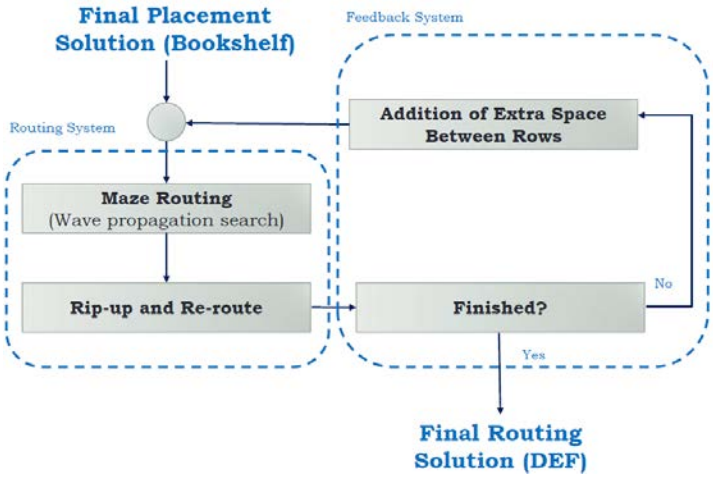

Figure 26. A feedback system design of the routing stage based on Qrouter.

To begin with, the parser aligns each cell within the specified routing grid defined by the pitch width in both x and y coordinates after receiving placement results. The extra space between rows and columns is introduced for more routing resources given the strictly constrained routing layers of current technology. The feedback system, written in Perl, introduces addition of extra space between rows uniformly once it receives the incomplete routing result generated by Qrouter. The temporary routing result determines the next modulation of the space corresponding to failed nets of the whole layout.

There are three modulation parameters of 7, 3, and 1 pitch size introduced between cells corresponding to the three possible outcomes determined by the magnitude of a failed net of 200 and 100 after each routing. A program written in Perl then performs the statistical analysis of complete or partial routing results for further comparisons such as average via counts and the longest routing path. This feedback routing system has accomplished 4-bit, 8-bit, 16-bit, 32-bit Kogge-Stone Adder and 8-bit, 16-bit Integer Divider with a H-Tree clock network topology. In summary, the following software developments were completed: (1) parsers, converting file formats required by different EDA tools (2) a feedback routing system, completing all net routings (3) a statistical analysis software, reporting layout results after the routing system.

3.4.3. Interconnect Delay and Frequency Modeling in RSFQ Circuits

Delay of signal propagation through a PTL line can be calculated as

$$t_{ptl} = \frac{PTL\ length\ (\mu m)}{100\ (\mu m/ps)} \qquad (1)$$

SFQ circuits can work properly with zero-skew clock distribution networks such as the one shown below. Consequently, a min clock cycle could be calculated as

$$t_{clk} \geq t_{c2q} + t_{tx} + t_{ptl} + t_{rx} + t_{setup} \qquad (2)$$

where t_{clk} represents min clock period, t_{c2q} and t_{tx} (t_{rx}) denote the Clock-to-Q delay of the source cell (~7ps) and PTL transmitter (receiver) delay (~2ps). t_{setup} account for the setup time of the destination cell (~3ps) and t_{ptl} , which denotes the PTL delay, can be calculated using equation 1.

Using the estimated values for t_{c2q}, t_{tx} (t_{rx}), and t_{setup} , the minimum clock period can be simplified as

$$t_{clk} \approx t_{ptl} + 14\,ps \qquad (3)$$

If the proposed HL-tree clock network is implemented, using a group size of k, the delay of linear clock propagation in cells of the same delay is added to the total clock cycle value. The minimum clock period for an HL-tree clock network is calculated as follows:

$$t_{clk} = t_{ptl} + 14ps + (k-1) \cdot t_{split} \qquad (4)$$

where t_{split} accounts for the delay of the splitter cell used for splitting the clock signal and passing that to the next cell in the same group. It is based on MIT-LL SFQEE5 process technology and is equal to 2 ps. Examples of H-tree and HL-tree clock routings (for the linear propagation portion of the clock tree) are shown in Figure 27 and Figure 28, respectively.

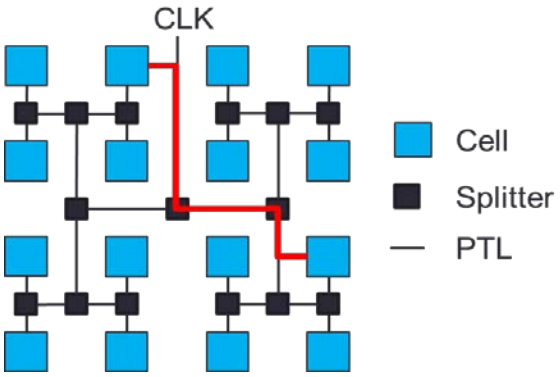

Figure 27. H-tree clock network.

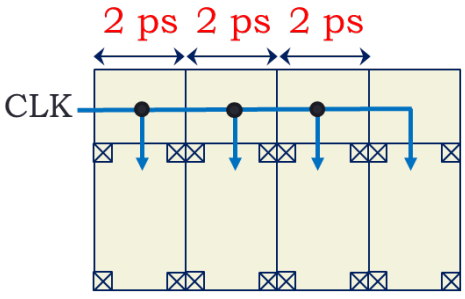

Figure 28. HL-tree clock propagation delay.

Using the HL-tree clock leads to a higher clock cycle and a lower clock frequency. However, it can decrease the total cell area significantly. With cell sizes based on the MIT-LL SFQEE5 process technology as shown in Figure 29, area savings for using HL-tree clock network are shown in Figure 30.

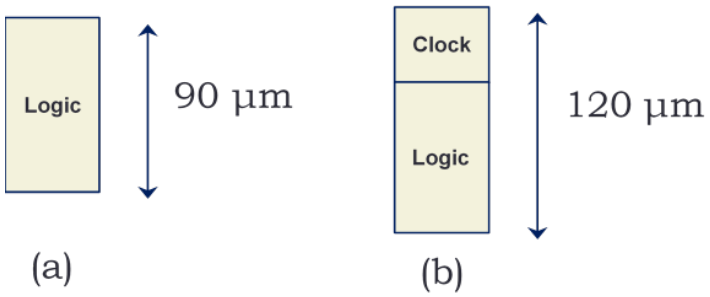

Figure 29. Cell Structure for (a) H-tree clock network (2) HL-tree clock network.

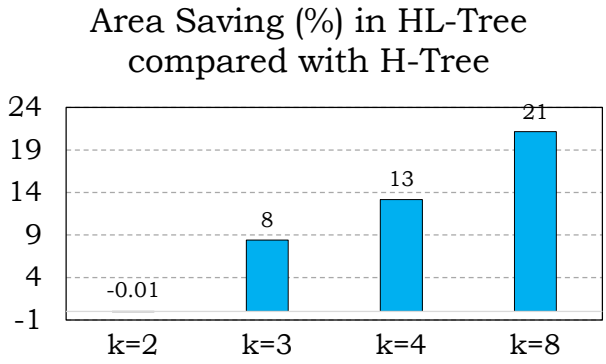

Figure 30. Percentage of area savings using HL-tree clock network with group size of k. Total number of 30 rows and 48 cells per row is assumed.

In the above example (c.f. Figure 30), it is assumed that there are a total number of 30 rows, and there are 48 cells in each row. Using a group size of k=4, the area is reduced by 13% while the minimum clock cycle increases by 6ps. In the above example, the delay of the PTL line (t_{ptl}) , which is typically equal to the half-perimeter of the chip, assuming average cell width of 40 μm and height of 120 μm, is equal to $\frac{30*120+40*2*48}{100} = 74.4\, ps$. Consequently, the clock frequency decreases by 8%, hence area*delay improves using the HL-tree clock network.

4. RESULTS AND DISCUSSION

To evaluate the methodologies and algorithms developed for RSFQ design, we have synthesized multiple arithmetic circuits as described in section 3.3, and performed placement, clock tree synthesis, and routing to generate the layout of these circuits. The post place and route results for some of these circuits are shown in Table 2.

Table 2. Post place and route results for some arithmetic circuits.

Circuit	4-bit KSA	8-bit KSA	16-bit KSA	32-bit KSA	8-bit Integer Divider	8-bit Integer Divider	16-bit Integer Divider
Clock Network	H-Tree	H-Tree	H-Tree	H-Tree	H-Tree	HL-Tree (k=2)	HL-Tree (k=2)
# PTL Layers	4	4	4	4	4	4	5
Logical Depth	6	8	10	13	84	84	300
Gate Count	87	230	627	1,645	2,159	2,159	15,583
PB DFFs	27	80	215	599	1,902	1,902	14,678
Avg. Via	2.2	2.3	2.3	2.4	2.5	2.9	3.1
Chip Area (mm^2)	**4.60**	**7.70**	**15.01**	**32.09**	**38.86**	**31.69**	**199.71**
Max PTL (μm)	1,553	1,740	2,790	4,170	4,905	4,790	13,860
Frequency (GHz)	**33.86**	**31.85**	**23.87**	**17.95**	**15.86**	**16.16**	**6.55**

KSA: Kogge-Stone Adder
PB: Path balancing

It can be seen that, using the proposed HL-tree clock network, the total area for the 8-bit integer divider decreases by 18% which proves the effectiveness of this approach.

As part of post place-and-route verification, we synthesized Verilog simulations directly from LEF/DEF post place-and-route file combinations. In this step, every component is linked to an equivalent simulation module for the component, as extracted with TimEx. Pulse transmission delay is modeled by calculating a delay time from the interconnect length, and inserting this delay between the output of a source component and the input of a target component. We use the open source HDL simulator iverilog for this project, but iverilog does not support wire delay assignments. An elegant work-around was thus developed that links every output to a wire, every input to a register, and then assigns the wire value to the register with the correct delay. This verification step validates the HDL cell library functionality as well as the synthesis, placement and routing steps.

As a demonstration, a 4-bit Kogge-Stone Adder with an H-Tree clock was extracted to a simulation model, and simulated successfully up to 50 GHz. The simulation includes accurate timing of all gates and wire delays, but does not include jitter.

Icarus Verilog was used to test the operation of the simulated model. Figure 31 shows the console output for a 4-bit Kogge-Stone adder as the Device Under Test (DUT) with a clock cycle of 1000 ps. It is observed in Figure 31 that the component operates as designed when driven by a 1 GHz frequency clock.

5. CONCLUSIONS

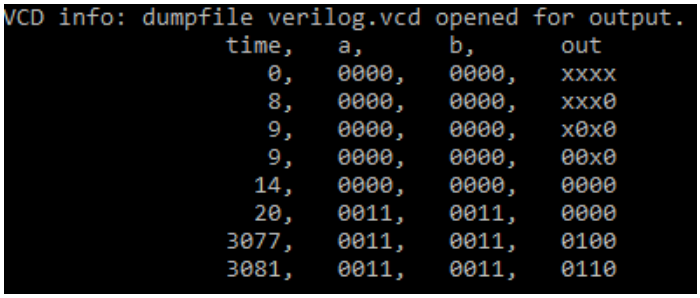

Figure 31. Verilog verification of the operation of a 4-bit Kogge-Stone adder at 1 GHz.

The effort of this Seedling project was to develop a prototype, open-source suite of tools for synthesis and physical design of RSFQ logic circuits. Throughout the process, a generic RSFQ cell library was developed with which to test the tool suite, and an automated HDL extraction tool was developed to find accurate timing models for all library cells. A Static Timing Analysis module was also developed to find the highest operating frequency without the use of pipelining, which presents the worst-case clock frequency for a circuit. Furthermore, we developed and implemented various methodologies for synthesis of RSFQ circuits, and integrated them into an open-source framework for logic synthesis (ABC [2]). Additionally, we developed a complete placement and clock tree synthesis flow to efficiently place large number of gates into the chip area, and create a clock network to propagate the clock signal to all the gates, while reducing the total wirelength and chip area. Finally, we utilized and open-source router tool (Qroute [9]) to perform the signal connections.

Publications Arising from This Seedling Project:

1. N. Katam and M. Pedram. “SFQ Circuit Design and Efficient Implementation of a 16-Bit Integer Divider Circuit,” draft, to be submitted to IEEE Trans. on Applied Superconductivity, 2017.
2. C. J. Fourie. “Flux Loop Analysis for SFQ Circuit Optimization and Automatic Extraction of Verilog Models,” draft, to be submitted to IEEE Trans. on Applied Superconductivity, 2017.
3. N. Katam, A. Shafaei, and M. Pedram. ”Design of Complex Rapid Single-Flux-Quantum Cells with Application to Logic Synthesis,” Intern. Superconductive Elec. Conf., Jun 2017.
4. J. A. Delport, and C. J. Fourie, “Static Timing Analysis for Pre- and Post-placed Superconducting Circuit Electronics,” Intern. Superconductive Elec. Conf., Jun 2017.

5. C. J. Fourie, “Flux Loop Analysis for RSFQ/ERSFQ Circuit Functionality Evaluation, Optimization and Timing Extraction.” Intern. Superconductive Elec. Conf., Jun 2017.
6. S. Nazar-shahsavani, T. R. Lin, A. Shafaei , C. J. Fourie and M. Pedram. "An Integrated Row-Based Cell Placement and Interconnect Synthesis Tool for Large SFQ Logic Circuits," IEEE Trans. on Applied Superconductivity, Vol. 27, No. 4, Mar. 2017.
7. N. Katam, A. Shafaei, and M. Pedram. ”Design of Multiple Fanout Clock Distribution Network for Rapid Single Flux Quantum Technology,” Proc. of Asia and South Pacific Design Automation Conf., Jan. 2017.
8. J. A. Delport, P. J. Peiser, N. Katam, M. Pedram, and C. J. Fourie, “SFQ logic cell library design for automated row-based layout,” IEEE Applied Superconductivity Conf., Denver, Colorado, 4-9 Sept. 2016.

References

[1] L. C. Müller and C. J. Fourie, "Automated state machine and timing characteristic extraction for RSFQ circuits," *IEEE Trans. Appl. Supercond.,* vol. 24, p. 1300110, 2014.

[2] "A System for Sequential Synthesis and Verification. [online] Available: https://people.eecs.berkeley.edu/~alanmi/abc/".

[3] "Berkeley Logic Interchange Format (BLIF). [online] Available : https://www.cse.iitb.ac.in/~supratik/courses/cs226/spr16/blif.pdf".

[4] A. S. a. M. P. N. Katam, "Design of Multiple Fanout Clock Distribution Network for Rapid Single Flux Quantum Technology," in *Proc. of Asia and South Pacific Design Automation Conf.*, 2017.

[5] D. L. a. I. L. M. MyungChul Kim, "SimPL: An Effective Placement Algorithm," 2011.

[6] U. S. a. F. M. J. P. Spindler, "Kraftwerk2 : a fast force-directed quadratic placement approach using an accurate net model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,,* Aug 2008.

[7] A. B. K. C.-K. K. a. C.-W. A. T. J. Cong, "Bounded-skew clock and steiner routing," *ACM Trans. Des. Autom. Electron. Syst,* 1998.

[8] "Bookshelf As a New Electronic Medium. [online] Available: http://vlsicad.eecs.umich.edu/BK/elemed.html".

[9] "Open Circuit Design. [online] Available:opencircuitdesign.com".

[10] H. -Y. C. a. Y.-W. Chang, "“Global and detailed routing,” Electronic Design Automation: Synthesis,," 2009.

[11] S. M. S. a. H. Youssef, "“Grid Routing,” VLSI Physical Design Automation: Theory and Practice,".

[12] C. Y. Lee, "An Algorithm for Path Connections and Its Applications".

List of Acronyms

ABC	Logic Synthesis tool name
BFS	Breadth First Search
BLIF	Berkeley Logic Interchange Format
BST/DME	A Clock tree synthesis tool
CLA	Carry Look-ahead Adder
CMOS	Complementary Metal Oxide Semiconductor
DC	Direct Current
DCSFQ	DC-to-SFQ Converter
DEF	Design Exchange Format
DFF	D Flip-Flop
DUT	Device Under Test
EDA	Electronic Design Automation
ERSFQ	Energy Efficient Single Flux Quantum
FSM	Full Subtractor with Multiplexer
GDS	Graphic Database System
HDL	Hardware Description Language
HL-tree	Hybrid clock tree with No skew and Linear
HPWL	Half-Perimeter Wirelength
HSM	Half Subtractor with Multiplexer
H-tree	No skew Clock distribution tree
I/O	Input /Output
JJ	Josephson Junction
JSIM	Josephson junction circuit Simulator
JTL	Josephson Transmission Line
LAL	Look Ahead Legalization
LEF	Liberty Exchange Format
MIT-LL	Massachusetts Institute of Technology- Lincoln Laboratory
NDRO	Non-Destructive Readout
PB	Path Balancing
PTL	Passive Transmission Line
Qrouter	Routing tool
RSFQ	Rapid Single Flux Quantum
SFQ	Single Flux Quantum
SimPL	Placement algorithm
STA	Static Timing Analysis
TimEx	Tool name
VLSI	Very Large Scale Integration
βC	Stewart-McCumber parameter